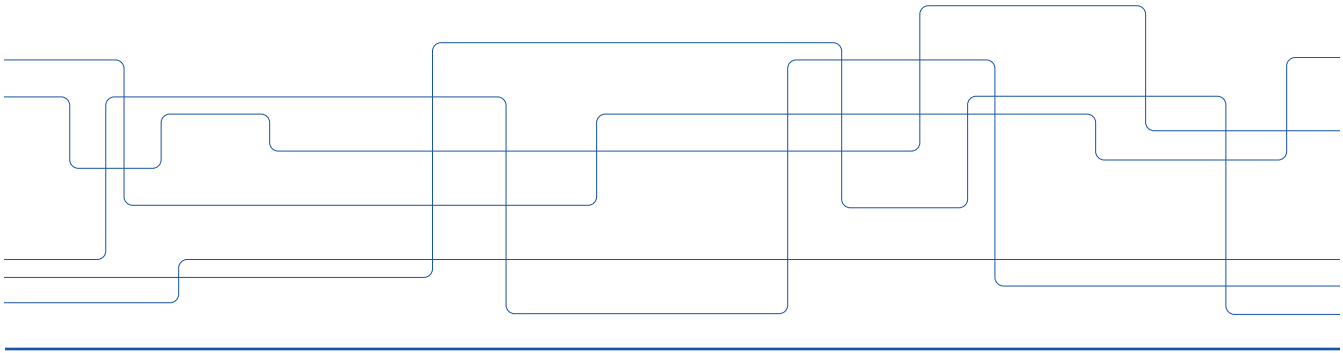
FORTRAN a crash course

Nek5000 specific FORTRAN features



Outlook

- Very short history of FORTRAN
- Basics of syntax
- Variables
- Loops
- Conditional statements
- Subprograms
- C/CUDA binding

Presentation goal

- This is just a short overview of Fortran (mainly Fortran 77) features that can be found in Nek5000 and is not meant to be a comprehensive description of the language. The main aim is to simplify reading of Nek5000 code showing not obvious tricks and possibly "confusing" code structures.
- Further reading:
  - Professional Programmer's Guide to Fortran77: https://www.star.le.ac.uk/~cgp/prof77.pdf
  - Fortran wiki : http://fortranwiki.org/

```
C
C     This next statement is to overcome the underflow bug in the i860.
C     It can be removed at a later date.  11 Aug 1990   pff.
C
      IF(ABS(Z) .LT. 1.0E-25) Z = 0.0
C
```

Very short history of FORTRAN

FORTRAN stands for *Formula Translation* and dates back to 1950s.

ebay.com

Fortran standards:

- Fortran 66
  - Thankfully largely obsolete
- Fortran 77
  - Mostly obsolete, but still in use
- Fortran 90
  - Significant modernization
- Others, not so important for this discussion
  - Fortran 95
  - Fortran 2003
  - Fortran 2008
  - Fortran 2018

Advantages of FORTAN

- Simple and easy to learn
- Fast
  - Compiles as well as language features that can inhibit performance are absent.
    - > C can give similar performance with care (e.g. need "restrict" keyword everywhere etc)
    - > Matlab/Python require extensive use of performance libraries/toolkits to get similar performance. (e.g. numpy).
  - Was designed from the beginning to rival hand written machine code for performance
  - Large set of optimised libraries.
- Good support for arrays and complex numbers
- Big number of available compilers (including free)
- Large amount of legacy code

Drawbacks of FORTAN

- Static memory allocation (Fortran 77)
- Very limited number of language feature (Fortran 77)
- Slow upgrade of legacy code to new standards

Basics of syntax

- Case insensitive:
  - Fortran does not consider case; e.g. "**DATA**", "**Data**" and "**data**" are all equivalent.
  - Exception is the case for text output.
- Comments:
  - Lines starting with **C** or **c** are treated as comment and ignored
  - Remaining part of line can be commented out using the **!** symbol

```
c              - Explicit treatment of the convection term.
c              - Velocity/stress formulation.

      call plan1 (igeom) ! The NEKTON "Classic".
```

Basics of syntax

Fixed format of Fortran 77**:**

- Column 1 for comment mark
- Columns 2-5 for numeric labels that can be referenced in a code (mainly used for old method of loops or for goto statements or for formatted output)
- Column 6 for continuation character (can be anything, shows this line is continuation of previous line)
- Code should be from 7 to 72
- Anything after column 72 ignored, most compilers accept a flag to extend to column 132, but best not used.
- Smart editors can help

Fortran 90 introduces **free format**; something more sensible, but not used by NEK5000

Basics of syntax

Fixed format example

```
c
c     Compute base flow rate
c
      if (icvflow.eq.1) base_flow = glsc2(vxc,bm1,ntot1)/domain_length
      if (icvflow.eq.2) base_flow = glsc2(vyc,bm1,ntot1)/domain_length
      if (icvflow.eq.3) base_flow = glsc2(vzc,bm1,ntot1)/domain_length
c
      if (nio.eq.0 .and. loglevel.gt.2) write(6,1)
     $   istep,chv(icvflow),base_flow,domain_length,flow_rate
    1    format(i11,'  basflow ',a1,11x,1p3e13.4)
```

Basics of syntax

- Specific parts of the code can be included/excluded from execution at a compilation step using **C-type pre-processing**.
- Pre-processing becomes important when binding Fortran 77 and C routines.

```
      if (iftran) call settime
      if (ifmhd ) call cfl_check
      call setsolv
      call comment

#ifdef CMTNEK
      if (nio.eq.0.and.istep.le.1) write(6,*) 'CMT branch active'
      call cmt_nek_advance
      return
#endif
```

Variables

```
real min_comm, max_comm, avg_comm

real comm_timers(8)
integer comm_counters(8)
character*132 s132
```

Basic variable types: **integer**, **real**, **complex**, **double precision**, **double complex**, **logical**, **character**

Implicit types:

- Comes from days of punch cards again where saving lines (card per line) was useful.
- Fortran has implicit typing based on variable name
  - Variables that start **I,J,K,L,M** are assumed **integers**
  - Other variables are assumed **real**.
- Can be overridden by explicit declaration.

Variables

```
real min_comm, max_comm, avg_comm

real comm_timers(8)
integer comm_counters(8)
character*132 s132
```

Implicit types:

- Explicit declaration required by:
  - Arrays
  - Other type variables, e.g.: **integer*8**, **real*16**, **logical**, **character**, **complex**
- Best practice now is to turn this off with **implicit none** statement.
  - This allows compiler to find your typos, as opposed to getting weird bugs at runtime because you had a typo in a variable name
  - Implicit types used thought NEK5000 (i.e. **implicit none** never used)

Real life example from NASA

- Implicit none turns on compiler checks, which can be very useful for finding bugs that can be simple for the compiler to find, but much harder for people
- Bug in the Project Mercury code (from NASA) where comma was replaced by period.
  - Change instruction from loop to assignment
    - Do 5 k=1,3
    - Do5k = 1.3
  - Spaces are ignored in the FORTRAN syntax
  - Implicit none would have flagged do5k as undefined variable
  - Using more modern loop syntax would have flagged syntax error in loop structure.
- See https://en.wikipedia.org/wiki/Mariner_1

Variables

Numerical precision:

- **real** equivalent to **real*4** – 32 bit
- **double precision** equivalent to **real*8** – 64 bit
- NEK5000 uses compiler flags (e.g. -r8 for Intel or PGI) to promote everything declared **real** to **real*8**.

```
*pgf*)          FCPP="-Mpreprocess"
                FR8="-r8"
                ;;
*gfortran*)     FCPP="-cpp"
                FR8="-fdefault-real-8 -fdefault-double-8"
                FF77="-std=legacy"
                ;;
*ftn*)          FCPP="-eZ"
                FR8="-sreal64"
                ;;
```

Variables

Arrays:

- Statically allocated in Fortran 77
- By default Fortran array indexing starts at 1
- Array range can be explicitly specified.
- Fortran arrays are stored in column-major order;
  - e.g. A(3,2) is stored A(1,1) A(2,1) A(3,1) A(1,2) A(2,2) A(3,2) A(1,3) A(2,3) A(3,3)
  - Important for loop execution and passing arguments to subroutines

```
subroutine hsmg_setup_fast1d(s,lam,nl,lbc,rbc,ll,lm,lr,ah,bh,n,ie)
integer nl,lbc,rbc,n
real s(nl,nl,2),lam(nl),ll,lm,lr
real ah(0:n,0:n),bh(0:n)

include 'SIZE'
parameter(lxm=lx1+2)
common /ctmp0/ b(2*lxm*lxm),w(2*lxm*lxm)
```

Variables

Global variables:

- Global variables are stored in named common blocks
  - Very simple structure containing just memory block starting and ending position (no content information)
  - Possible use for scratch space (extensively used in Nek5000)
  - Must be declared consistently in all subroutines/functions accessing a variable
  - Simplest if all are same type (prevents alignment/performance issue)
  - Can be declared in external file and included in a subroutine

```
subroutine hsmg_setup_fast1d(s,lam,nl,lbc,rbc,ll,lm,lr,ah,bh,n,ie)
integer nl,lbc,rbc,n
real s(nl,nl,2),lam(nl),ll,lm,lr
real ah(0:n,0:n),bh(0:n)

include 'SIZE'
parameter(lxm=lx1+2)
common /ctmp0/ b(2*lxm*lxm),w(2*lxm*lxm)
```

Variables

Global variables:

- Global variables are stored in named common blocks
  - Error prone
  - Little control over loaded variables
  - Replaced in Fortran 90 by modules
- **Include files and common blocks are extensively used in Nek5000 providing access to main parameters and global variables**

```
subroutine hsmg_setup_fast1d(s,lam,nl,lbc,rbc,ll,lm,lr,ah,bh,n,ie)
integer nl,lbc,rbc,n
real s(nl,nl,2),lam(nl),ll,lm,lr
real ah(0:n,0:n),bh(0:n)

include 'SIZE'
parameter(lxm=lx1+2)
common /ctmp0/ b(2*lxm*lxm),w(2*lxm*lxm)
```

Variables

```
      subroutine userf  (ix,iy,iz,ieg)
      include 'SIZE'              ! NX1, NY1, NZ1, NELV, NID
      include 'NEKUSE'            ! FFX, FFY, FFZ
      include 'PARALLEL'          ! GLLEL
      include 'INPUT'             ! IF3D
      include 'SFD'               ! IFSFD, SFDCHI, BFS?c


      integer iel

      iel = gllel(ieg)

!     SFD
      if (IFSFD) then
         FFX = FFX - SFDCHI*BFSX(ix,iy,iz,iel)
         FFY = FFY - SFDCHI*BFSY(ix,iy,iz,iel)
         if (IF3D) FFZ = FFZ - SFDCHI*BFSZ(ix,iy,iz,iel)
      else
         FFX = 0.0
         FFY = 0.0
         if (IF3D) FFZ = 0.0
      endif

      return
      end
```

Variables

Example of include file **SIZE**

```
! BASIC
parameter (ldim=3)                ! domain dimension (2 or 3)
parameter (lx1=8)                 ! GLL points per element along each direct
parameter (lxd=12)                ! GL  points for over-integration (dealias
parameter (lx2=lx1-0)             ! GLL points for pressure (lx1 or lx1-2)

parameter (lelg=1000)             ! max number of global elements
parameter (lpmin=1)               ! min number of MPI ranks
parameter (lelt=lelg/lpmin + 3)   ! max number of local elements per MPI ran
parameter (ldimt=1)               ! max auxiliary fields (temperature + scal

! OPTIONAL
parameter (ldimt_proj=1)          ! max auxiliary fields residual projection
parameter (lelr=lelt)             ! max number of local elements per restart
parameter (lhis=1)                ! max history/monitoring points
parameter (maxobj=1)              ! max number of objects
parameter (lpert=1)               ! max number of perturbations
```

Variables

Example of include file **SOLN**

```
c     Solution data
      real vx     (lx1,ly1,lz1,lelv)
     $    ,vy     (lx1,ly1,lz1,lelv)
     $    ,vz     (lx1,ly1,lz1,lelv)
     $    ,vx_e   (lx1,ly1,lz1,lelv)
     $    ,vy_e   (lx1,ly1,lz1,lelv)
     $    ,vz_e   (lx1,ly1,lz1,lelv)
     $    ,t      (lx1,ly1,lz1,lelt,ldimt)
     $    ,vtrans (lx1,ly1,lz1,lelt,ldimt1)
     $    ,vdiff  (lx1,ly1,lz1,lelt,ldimt1)
     $    ,bfx    (lx1,ly1,lz1,lelv)
     $    ,bfy    (lx1,ly1,lz1,lelv)
     $    ,bfz    (lx1,ly1,lz1,lelv)
     $    ,cflf   (lx1,ly1,lz1,lelv)
     $    ,bmnv   (lx1*ly1*lz1*lelv*ldim,lorder+1) ! binv*mask
     $    ,bmass  (lx1*ly1*lz1*lelv*ldim,lorder+1) ! bmass
     $    ,bdivw  (lx1*ly1*lz1*lelv*ldim,lorder+1) ! bdivw*mask
     $    ,c_vx   (lxd*lyd*lzd*lelv*ldim,lorder+1) ! characteristics
     $    ,fw     (2*ldim,lelt)                    ! face weights for DG

      common /vptsol/ vxlag, vylag, vzlag, tlag, vgradt1, vgradt2,
     $     abx1, aby1, abz1, abx2, aby2, abz2, vdiff e,
```

Loops

Two possible variants of loop construction can be found in Nek500:

- old **do (s) i=start, end [,stride]**

```
      DO 1001 IE=1,NELT
         IF (.NOT.IFDFRM(IE)) THEN
            DO 1000 IZ=1,lz1
            DO 1000 IY=1,ly1
            DO 1000 IX=1,lx1
               G4M1(IX,IY,IZ,IE)=G1M1(IX,IY,IZ,IE)/WXM1(IX)
               G5M1(IX,IY,IZ,IE)=G2M1(IX,IY,IZ,IE)/WYM1(IY)
               G6M1(IX,IY,IZ,IE)=G3M1(IX,IY,IZ,IE)/WZM1(IZ)
1000        CONTINUE
         ENDIF
1001  CONTINUE
```

- more modern

```
      l=0
      do k=1,nz
      do j=1,ny
         wjk=wi(j)*wi(k)
         do i=1,nx
            l=l+1
            wk(l) = wjk*wi(i)
         enddo
      enddo
      enddo
```

Loops

Implicit index merging; notice different shapes of arrays and loop bounds:

- jacmi(lx1*ly1*lz1,lelt); sij(lx1*ly1*lz1,6,lelv); rxm1(lx1,ly1,lz1,lelt)
- nxyz = lx1*ly1*lz1

```
      do e=1,nelv
         call local_grad2(ur,us,u,N,e,dxm1,dxtm1)
         call local_grad2(vr,vs,v,N,e,dxm1,dxtm1)

         do i=1,nxyz
            j = jacmi(i,e)

            sij(i,1,e) = j*  ! du/dx + du/dx
     $        2*(ur(i)*rxm1(i,1,1,e)+us(i)*sxm1(i,1,1,e))

            sij(i,2,e) = j*  ! dv/dy + dv/dy
     $        2*(vr(i)*rym1(i,1,1,e)+vs(i)*sym1(i,1,1,e))

            sij(i,3,e) = j*  ! du/dy + dv/dx
     $        (ur(i)*rym1(i,1,1,e)+us(i)*sym1(i,1,1,e) +
     $         vr(i)*rxm1(i,1,1,e)+vs(i)*sxm1(i,1,1,e) )

         enddo
      enddo
```

Loops

- Loop breaking can be performed with **go to** statement transferring control to the labelled executable statement.
- **go to** is simple to use and allows to write a shorter code, but could make it hard to read
- Replaced in Fortran 90 with **exit** and **cycle** statements
- In some cases **go to** is overused in Nek5000.

```
      nfldt = 1+npscal
      do ifld=1,nfldt
         ifield = ifld + 1
         if (.not. iffilter(ifield)) goto 10
         call filterq(t(1,1,1,1,ifld),intv,
     $                lx1,lz1,wk1,wk2,intt,if3d,tmax(ifld))
         if (ifpert) then
            do j=1,npert
               call filterq(tp(1,j,1),intv,lx1,lz1,wk1,wk2,intt,if3d,
     $                      wmax)
            enddo
         endif
   10    mmax = mmax+1
c         omax(mmax) = glmax(tmax(ifld),1)
      enddo
```

Conditional statements

- Generic if statement

  if (logical expression) executable expression

- Logical operators: **.not., .and., .or., .xor.**

x**.gt.**y	x>y	x**.ge.**y	x≥y	x**.eq.**y	x=y
x**.lt.**y	x<y	x**.le.**y	x≤y	x**.ne.**y	x≠y

```
if (icvflow.eq.1) then
   call cdtp     (respr,v1mask,rxm2,sxm2,txm2,1)
elseif (icvflow.eq.2) then
   call cdtp     (respr,v2mask,rxm2,sxm2,txm2,1)
else
   call cdtp     (respr,v3mask,rxm2,sxm2,txm2,1)
endif
```

Subprograms

- Subroutines (no return value)

```
      subroutine hsmg_setup_intpm(jh,zf,zc,nf,nc)
      integer nf,nc
      real jh(nf,nc),zf(1),zc(1)
      include 'SIZE'
      real w(2*lx1+2)
      do i=1,nf
         call fd_weights_full(zf(i),zc,nc-1,1,w)
         do j=1,nc
            jh(i,j)=w(j)
         enddo
      enddo
      return
      end
```

```
         call hsmg_setup_intpm(
     $           mg_jh(1,l),mg_zh(1,l+1),mg_zh(1,l),nf,nc)
```

Subprograms

- Functions return a single value
- Name must correspond to implicit types or should be explicitely declared

```
integer function log2(k)
RK=(K)
RLOG=LOG10(RK)
RLOG2=LOG10(2.0)
RLOG=RLOG/RLOG2+0.5
LOG2=INT(RLOG)
return
END
```

```
log_np=log2(np)
np2 = 2**log_np
if (np2.eq.np) call gp2_test(ivb)    ! a
```

Subprograms

- No checking of subroutine/function prototypes is done by default i.e. compiler will not tell you if you make a mistake with variables in call statement
- All arguments to subroutines/functions are transferred as pointers (call by reference)
  - Use of single variable as multiple arguments in a single call is not allowed (important for optimisation)
  - Possible array reshaping assuming continuous set of data

```
      subroutine invers2(a,b,n)
      REAL A(1),B(1)
C
      include 'OPCTR'
C
      DO 100 I=1,N
         A(I)=1./B(I)
  100 CONTINUE
      return
      END
```

Subprograms

- Subroutine arguments called by reference

```
      subroutine hsmg_setup_intpm(jh,zf,zc,nf,nc)
      integer nf,nc
      real jh(nf,nc),zf(1),zc(1)
      include 'SIZE'
      real w(2*lx1+2)
      do i=1,nf
         call fd_weights_full(zf(i),zc,nc-1,1,w)
         do j=1,nc
            jh(i,j)=w(j)
         enddo
      enddo
      return
      end
```

```
         call hsmg_setup_intpm(
     $           mg_jh(1,l),mg_zh(1,l+1),mg_zh(1,l),nf,nc)
```

Formatted Output

- Formatted output is done using write and print statements.
  - Write takes two arguments followed by list of things to output.
    - > The first argument is normally 6 for standard output (i.e. output to screen)
    - > The second is the label of the format statement.
    - > Write(*,*) will give unformatted output, which is useful for quick tests.

```
c
c     Compute base flow rate
c
      if (icvflow.eq.1) base_flow = glsc2(vxc,bm1,ntot1)/domain_length
      if (icvflow.eq.2) base_flow = glsc2(vyc,bm1,ntot1)/domain_length
      if (icvflow.eq.3) base_flow = glsc2(vzc,bm1,ntot1)/domain_length
c
      if (nio.eq.0 .and. loglevel.gt.2) write(6,1)
     $   istep,chv(icvflow),base_flow,domain_length,flow_rate
    1   format(i11,'  basflow ',a1,11x,1p3e13.4)
```

Format Statement

- Format statement requires label so it can be referenced.
  - Label must be unique in the scope
  - Format statement determines how numbers are output
  - If given format cannot contain the number to be output you will get asterisk output (******) e.g. if you give 4 digits for integer output and the output is 5.

```
c
c     Compute base flow rate
c
      if (icvflow.eq.1) base_flow = glsc2(vxc,bm1,ntot1)/domain_length
      if (icvflow.eq.2) base_flow = glsc2(vyc,bm1,ntot1)/domain_length
      if (icvflow.eq.3) base_flow = glsc2(vzc,bm1,ntot1)/domain_length
c
      if (nio.eq.0 .and. loglevel.gt.2) write(6,1)
     $   istep,chv(icvflow),base_flow,domain_length,flow_rate
    1    format(i11,'  basflow ',a1,11x,1p3e13.4)
```

Calling C/CUDA routines from Fortran

- Nek5000 includes some C and CUDA routines, called from Fortran
- Fortran name mangling
  - Most common now is adding single underscore and with lower case subroutine names (historically others existed)
    - > Subroutine abc becomes routine abc_ internally in compiler.
  - Pre-processor is used to control mangling method, example code in C files.
    - > Require "extern C" to make sure C compiler does not do any name mangling as well (or more likely C++/CUDA compiler for function overloading etc).
  - Variables are passed by reference in FORTRAN
  - Passing character arrays somewhat complicated, as FORTRAN character arrays also have length information.
  - Note data layout of arrays is reversed
    - >Fortran fastest moving index is first index A(i,j)
    - >In C fastest moving index is second index A[j][i]